Data Importing and Exporting

Contents

1 Review

So far we have:

- learned how to load in packages
- learned how to create some plots with built in datasets
- Some basics of coding
- Basics of naming objects
- General principals of calling functions.
- Transform data

This week we will learn:

- How to read in .csv files.
- How to read in excel and other data types.
- How to export data.

Some Important terms:

- Header - When importing a file into `R`, users need to tell the software whether a header is present so that it knows whether to treat the 1st line as variable names or observed data values.
- Delimiter - The delimiter is a character used to separate the entries in each line. This tells `R` when a specific entry begins and ends.
- Missing value - This is used to denote a missing value. When reading the file, R will turn these entries into the form it recognizes: NA.

2 Introduction

```
library(tidyverse)
library(readxl)
library(haven)
```

To start we'll learn how to read in data using the `readr` package. Some of the most common functions in `readr` are:

- `read_csv()` reads comma delimited files, `read_csv2()` reads semicolon separated files (common in countries where `,` is used as the decimal place), `read_tsv()` reads tab delimited files, and `read_delim()` reads in files with any delimiter.
- `read_fwf()` reads fixed width files. You can specify fields either by their widths with `fwf_widths()` or their position with `fwf_positions()`. `read_table()` reads a common variation of fixed width files where columns are separated by white space.

These functions all have similar syntax: once you've mastered one, you can use the others with ease.

The focus here is on `read_csv()` once you understand this the others are straightforward.

The other packages we'll use are `readxl`, which we'll use to read in excel files, and `haven` which can be used to read in other common data types. Please ensure you have these packages installed before you commence.

3 Filepath and working directory

To read in the data, we need the full filepath. On my computer, the full filepath for this variable is:

*"/Users/mclainfamily/Library/CloudStorage/OneDrive-UniversityofSouthCarolina/

Teaching/711_Fall_2022/Notes/data/students.csv"*

which is a bunch to type in. You can type that in if you'd like. Below we'll explore option of simplifying this.

The working directory is the location that `R` will look to when the full filepath is not given. You can see what your working directory is by using `getwd()`

```
getwd()
```

```
## [1] "/Users/mclainfamily/Library/CloudStorage/OneDrive-UniversityofSouthCarolina/Teaching/711_Fall_2
```

Notice that the data are in a subfolder of my working directory. As a result, for the filepath I only need to use

"data/students.csv"

3.1 Setting working directory

If you need to change your working directory, you use `setwd()`

```
setwd("/Users/mclainfamily/Library/CloudStorage/OneDrive-UniversityofSouthCarolina/
      Teaching/711_Fall_2022/Notes")
```

How to get the filepath? On Windows you can get it from your windows file explorer. **However,** you will need to change the \ on Windows so they look like:

```
C:\\Users\\mclaina\\OneDrive - University of South Carolina\\Teaching\\711_Fall_2022\\Notes
```

To get the filepath on mac, right-click the desired folder, then hold down the 'option' key and click 'Copy "Notes" as Pathname'.

You can also set your working directory in RStudio by:

- going to “Files” in the Helper pane ->
- navigating to where you want your working directory to be ->
- Click “More” ->
- Click “Set as Working Directory”.

4 Reading in a .csv file with ‘readr‘

First, let’s look at what a .csv file looks like.....

Now, let’s read in a .csv file

```
students <- read_csv("data/students.csv")
```

```
## Rows: 6 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): Full Name, favourite.food, mealPlan, AGE
## dbl (1): Student ID
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

When you run `read_csv()` it prints out a message that tells you how many rows (excluding the header row) and columns the data has along with the delimiter used, and the column specifications (names of columns organized by the type of data the column contains). It also prints out some information about how to retrieve the full column specification as well as how to quiet this message.

```
students
```

```
## # A tibble: 6 x 5
##   `Student ID` `Full Name`      favourite.food     mealPlan            AGE
##          <dbl> <chr>            <chr>              <chr>               <chr>
## 1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2            2 Barclay Lynn     French fries       Lunch only          5
## 3            3 Jayendra Lyne    N/A                Breakfast and lunch 7
## 4            4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6            6 Güvenç Attila    Ice cream          Lunch only          6
```

You can also supply an inline csv file. This isn’t really useful for creating data, but it will be useful to demonstrate how the program works.

```
read_csv("a,b,c
1,2,3
4,5,6")
```

```
## # A tibble: 2 x 3
##       a     b     c
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

In both cases `read_csv()` uses the first line of the data for the column names, which is a very common convention. There are two cases where you might want to tweak this behavior:

1. Sometimes there are a few lines of metadata at the top of the file. You can use `skip = n` to skip the first `n` lines; or use `comment = "#"` to drop all lines that start with (e.g.) `#`.

```r
read_csv("The first line of metadata
  The second line of metadata
  x,y,z
  1,2,3", skip = 2)
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

```r
read_csv("# A comment I want to skip
  x,y,z
  1,2,3", comment = "#")
```

```
## # A tibble: 1 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
```

2. The data might not have column names. You can use `col_names = FALSE` to tell `read_csv()` not to treat the first row as headings, and instead label them sequentially from `X1` to `Xn`:

```r
read_csv("1,2,3
         4,5,6", col_names = FALSE)
```

```
## # A tibble: 2 x 3
##      X1    X2    X3
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```

Alternatively you can pass `col_names` a character vector which will be used as the column names:

````
```r
read_csv("1,2,3
         4,5,6", col_names = c("x", "y", "z"))
```

```
## # A tibble: 2 x 3
##       x     y     z
##   <dbl> <dbl> <dbl>
## 1     1     2     3
## 2     4     5     6
```
````

Another option that commonly needs tweaking is `na`: this specifies the value (or values) that are used to represent missing values in your file:

```r
read_csv("a,b,c
         1,2,.", na = ".")
```

```
## # A tibble: 1 x 3
##       a     b c
##   <dbl> <dbl> <lgl>
## 1     1     2 NA
```

This is all you need to know to read ~75% of CSV files that you'll encounter in practice. You can also easily

adapt what you've learned to read tab separated files with `read_tsv()` and fixed width files with `read_fwf()`. To read in more challenging files, you'll need to learn more about how readr parses each column, turning them into R vectors.

4.1 First steps

Let's take another look at the `students` data. In the `favourite.food` column, there are a bunch of food items and then the character string `N/A`, which should have been an real `NA` that R will recognize as "not available". This is something we can address using the `na` argument.

```
students <- read_csv("data/students.csv", na = c("N/A", ""))
students
```

```
## # A tibble: 6 x 5
##   `Student ID` `Full Name`      favourite.food     mealPlan            AGE
##          <dbl> <chr>            <chr>              <chr>               <chr>
## 1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2            2 Barclay Lynn     French fries       Lunch only          5
## 3            3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4            4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6            6 Güvenç Attila    Ice cream          Lunch only          6
```

Once you read data in, the first step usually involves transforming it in some way to make it easier to work with in the rest of your analysis.

For starters, notice the column names in the `students` file we read in are formatted in non-standard ways. In particular, the names `Student ID` and `Full Name` are a little problematic in `R` due to the space. Mostly, we want column names not to have a space.

```
students |> select(Full Name)
# Error: unexpected symbol in "students |> select(Full Name"
```

Gives an error. I would need to use:

```
students |> select('Full Name')
```

```
## # A tibble: 6 x 1
##   `Full Name`
##   <chr>
## 1 Sunil Huffmann
## 2 Barclay Lynn
## 3 Jayendra Lyne
## 4 Leon Rossini
## 5 Chidiegwu Dunkel
## 6 Güvenç Attila
```

As a result, it's nice to have variable names do not have spaces.

You might consider renaming them one by one with `dplyr::rename()`:

```
students |> rename(student_id = 'Student ID', full_name = 'Full Name')
```

```
## # A tibble: 6 x 5
##   student_id full_name        favourite.food     mealPlan            AGE
##        <dbl> <chr>            <chr>              <chr>               <chr>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2          2 Barclay Lynn     French fries       Lunch only          5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
```

```
## 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

Another option is to use the `janitor::clean_names()` function and turn all names into snake case at once. This function takes in a data frame and returns a data frame with variable names converted to snake case.

```
library(janitor)
students |>
  clean_names()
```

```
## # A tibble: 6 x 5
##   student_id full_name        favourite_food     meal_plan           age
##        <dbl> <chr>            <chr>              <chr>               <chr>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2          2 Barclay Lynn     French fries       Lunch only          5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

Another common task after reading in data is to consider variable types. For example, `meal_type` is a categorical variable with a known set of possible values. In R, factors can be used to work with categorical variables. We can convert this variable to a factor using the `factor()` function.

```
students <- students |>
  clean_names() |>
  mutate(meal_plan = factor(meal_plan))
students
```

```
## # A tibble: 6 x 5
##   student_id full_name        favourite_food     meal_plan           age
##        <dbl> <chr>            <chr>              <fct>               <chr>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2          2 Barclay Lynn     French fries       Lunch only          5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

Note that the values in the `meal_type` variable has stayed exactly the same, but the type of variable denoted underneath the variable name has changed from character (`<chr>`) to factor (`<fct>`).

Before you move on to analyzing these data, you'll probably want to fix the `age` column as well: currently it's a character variable because of the one observation that is typed out as `five` instead of a numeric `5`. We discuss the details of fixing this issue later in the course.

4.2 Other options for .csv files

The `read.csv()` is similar to `read_csv()` and is included in the base version of R. There are a few good reasons to favor readr functions over the base equivalents:

- They are typically much faster (~10x) than their base equivalents. Long running jobs have a progress bar, so you can see what's happening. If you're looking for raw speed, try `data.table::fread()`.
- They produce tibbles, and they don't use row names or munge the column names. These are common sources of frustration with the base R functions.

- They are more reproducible. Base R functions inherit some behavior from your operating system and environment variables, so import code that works on your computer might not work on someone else's.

5 Writing to a file

readr also comes with two useful functions for writing data back to disk: `write_csv()` and `write_tsv()`. Both functions increase the chances of the output file being read back in correctly by:

- Always encoding strings in UTF-8.
- Saving dates and date-times in ISO8601 format so they are easily parsed elsewhere.

If you want to export a csv file to Excel, use `write_excel_csv()` — this writes a special character (a "byte order mark") at the start of the file which tells Excel that you're using the UTF-8 encoding.

The most important arguments are `x` (the data frame to save), and `file` (the location to save it). You can also specify how missing values are written with `na`, and if you want to `append` to an existing file.

```
write_csv(students, "students-2.csv")
```

Now let's read that csv file back in. Note that the type information is lost when you save to csv:

```
students
```

```
## # A tibble: 6 x 5
##   student_id full_name        favourite_food     meal_plan           age
##        <dbl> <chr>            <chr>              <fct>               <chr>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2          2 Barclay Lynn     French fries       Lunch only          5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

```
write_csv(students, "students-2.csv")
read_csv("students-2.csv")
```

```
## # A tibble: 6 x 5
##   student_id full_name        favourite_food     meal_plan           age
##        <dbl> <chr>            <chr>              <chr>               <chr>
## 1          1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2          2 Barclay Lynn     French fries       Lunch only          5
## 3          3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4          4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5          5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6          6 Güvenç Attila    Ice cream          Lunch only          6
```

6 Reading in Excel files

Any excel file can be saved as a .csv, however, sometimes it is easier to read in an excel file instead. To do this we can use the `readxl` package and the `read_excel` function.

```
library(readxl)
data_xl <- read_excel("data/xlsx_test.xlsx", sheet = "data")
dict_xl <- read_excel("data/xlsx_test.xlsx", sheet = "dictionary")
data_xl
```

```
## # A tibble: 500 x 141
```

```
##    StartDate           EndDate             Status IPAddress      Progr~1 Durat~2
##    <dttm>              <dttm>              <chr>  <chr>            <dbl>   <dbl>
##  1 2021-11-18 13:05:30 2021-11-18 13:24:36 0      129.252.33.116     100    1146
##  2 2021-11-18 13:25:11 2021-11-18 13:28:13 0      129.252.33.116     100     182
##  3 2021-11-18 13:28:38 2021-11-18 13:30:46 0      129.252.33.116     100     128
##  4 2021-11-18 13:31:12 2021-11-18 13:33:17 0      129.252.33.116     100     125
##  5 2021-11-18 13:33:46 2021-11-18 13:35:52 0      129.252.33.116     100     126
##  6 2021-11-18 13:36:47 2021-11-18 13:39:37 0      129.252.33.116     100     169
##  7 2021-11-18 13:41:03 2021-11-18 13:43:34 0      129.252.33.116     100     151
##  8 2021-11-18 13:44:01 2021-11-18 13:46:42 0      129.252.33.116     100     160
##  9 2021-11-18 13:47:21 2021-11-18 13:50:01 0      129.252.33.116     100     159
## 10 2021-11-18 13:50:31 2021-11-18 13:52:57 0      129.252.33.116     100     146
## # ... with 490 more rows, 135 more variables: Finished <chr>,
## #   RecordedDate <dttm>, ResponseId <chr>, RecipientLastName <lgl>,
## #   RecipientFirstName <lgl>, RecipientEmail <lgl>, ExternalReference <lgl>,
## #   LocationLatitude <chr>, LocationLongitude <chr>, DistributionChannel <chr>,
## #   UserLanguage <chr>, id <chr>, cohort <chr>, ssis1_oret_base <chr>,
## #   ssis2_oret_base <chr>, ssis3_oret_base <chr>, ssis4_oret_base <chr>,
## #   ssis5_oret_base <chr>, ssis6_oret_base <chr>, ssis7_oret_base <chr>, ...
## # i Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

```
dict_xl
```

```
## # A tibble: 284 x 4
##    variable  value label              meta
##    <chr>     <dbl> <chr>              <chr>
##  1 StartDate    NA Start Date         varlab
##  2 EndDate      NA End Date           varlab
##  3 Status       NA Response Type      varlab
##  4 Status        0 IP Address         <NA>
##  5 Status        1 Survey Preview     <NA>
##  6 Status        2 Survey Test        <NA>
##  7 Status        4 Imported           <NA>
##  8 Status        8 Spam               <NA>
##  9 Status        9 Survey Preview Spam <NA>
## 10 Status       12 Imported Spam      <NA>
## # ... with 274 more rows
## # i Use `print(n = ...)` to see more rows
```

This function has similar arguments that can be used to:

- include the top row as the variable names (`col_names=TRUE`) or not (`col_names=FALSE`)
- specify NA values (`na="."`)
- skip the first n rows of data (`skip=n`)

There is another package `openxlsx` that allows you to read and write excel files in `R`.
I usually write files to csv documents

7 Reading in other formats

Another useful package is `haven` which allows you to read in files from SAS (`read_sas()`), Stata (`read_dta`), and SPSS (`read_sav`). These files have similar options to `read_csv` as they were created by the same developer team.

As an example, we'll read in a spss file as these files have some nice properties to them.

```
library(haven)
test_spss <- read_sav("data/spss_file.sav")
test_spss
```

```
## # A tibble: 500 x 141
##    StartDate           EndDate                    Status IPAdd~1 Progr~2 Durat~3
##    <dttm>              <dttm>                   <dbl+lbl> <chr>     <dbl>   <dbl>
##  1 2021-11-18 13:05:30 2021-11-18 13:24:36 0 [IP Addres~ 129.25~     100    1146
##  2 2021-11-18 13:25:11 2021-11-18 13:28:13 0 [IP Addres~ 129.25~     100     182
##  3 2021-11-18 13:28:38 2021-11-18 13:30:46 0 [IP Addres~ 129.25~     100     128
##  4 2021-11-18 13:31:12 2021-11-18 13:33:17 0 [IP Addres~ 129.25~     100     125
##  5 2021-11-18 13:33:46 2021-11-18 13:35:52 0 [IP Addres~ 129.25~     100     126
##  6 2021-11-18 13:36:47 2021-11-18 13:39:37 0 [IP Addres~ 129.25~     100     169
##  7 2021-11-18 13:41:03 2021-11-18 13:43:34 0 [IP Addres~ 129.25~     100     151
##  8 2021-11-18 13:44:01 2021-11-18 13:46:42 0 [IP Addres~ 129.25~     100     160
##  9 2021-11-18 13:47:21 2021-11-18 13:50:01 0 [IP Addres~ 129.25~     100     159
## 10 2021-11-18 13:50:31 2021-11-18 13:52:57 0 [IP Addres~ 129.25~     100     146
## # ... with 490 more rows, 135 more variables: Finished <dbl+lbl>,
## #   RecordedDate <dttm>, ResponseId <chr>, RecipientLastName <chr>,
## #   RecipientFirstName <chr>, RecipientEmail <chr>, ExternalReference <chr>,
## #   LocationLatitude <chr>, LocationLongitude <chr>, DistributionChannel <chr>,
## #   UserLanguage <chr>, id <chr>, cohort <dbl+lbl>, ssis1_oret_base <dbl+lbl>,
## #   ssis2_oret_base <dbl+lbl>, ssis3_oret_base <dbl+lbl>,
## #   ssis4_oret_base <dbl+lbl>, ssis5_oret_base <dbl+lbl>, ...
## # i Use `print(n = ...)` to see more rows, and `colnames()` to see all variable names
```

Notice the variable types. `POSIXct` is a Date-time format. We can dig in further, and see that the each variable has a label, and label values.

```
test_spss$cohort
```

```
## <labelled<double>[500]>: Cohort 1 or 2?
##   [1] 2 2 1 2 1 2 1 2 2 2 1 2 1 2 2 1 1 2 2 1 2 2 2 2 2 2 2 2 2 1 2 2 1 2 2 2
##  [38] 2 1 1 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 1 2 1 2 2 2 2 2 2 2 1 2 2 2 2
##  [75] 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 1 1 2 1 2 2 1 2 1 2 2 2 2 2 2 2 2 2 1 2 2
## [112] 2 2 2 2 2 1 2 1 2 2 1 1 2 2 2 1 2 2 2 2 2 1 1 1 1 2 2 2 2 1 2 2 2 1 2 2 2
## [149] 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 1 2 1 2 2 1 2
## [186] 1 2 2 2 2 2 1 2 2 2 2 2 2 2 1 1 2 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2
## [223] 2 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 1 1 2
## [260] 2 2 2 2 2 2 2 2 1 1 1 1 2 2 2 2 2 2 2 1 2 2 2 1 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2
## [297] 1 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 1 2 2 2 2 2 1 2 1 2 2 2 2 2 1 2 1 2 2 2 2 2
## [334] 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 1 2 1 2 2 2 2 2 2 2 2 1 2 2 2 1 2 1 2 1
## [371] 1 2 2 2 2 2 2 2 2 2 2 1 1 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 2 1 2 1 2 2 2
## [408] 1 1 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 1
## [445] 2 2 2 2 1 2 2 2 1 2 2 1 1 1 2 2 1 2 2 2 2 2 2 1 2 2 2 2 1 2 2 2 2 2 2 1 2 2 2 1 2 1 2 2 2
## [482] 2 2 1 1 2 2 2 2 2 2 2 2 1 2 1 2 2 2 2 2 2
##
## Labels:
##  value                            label
##      1         Cohort 1 [ID's 100-599]
##      2 Cohort 2 [ID's 600 and greater]
```

```
test_spss |>
  select(cohort)
```

```
## # A tibble: 500 x 1
```

```
##                                cohort
##                               <dbl+lbl>
##  1 2 [Cohort 2 [ID's 600 and greater]]
##  2 2 [Cohort 2 [ID's 600 and greater]]
##  3 1 [Cohort 1 [ID's 100-599]]
##  4 2 [Cohort 2 [ID's 600 and greater]]
##  5 1 [Cohort 1 [ID's 100-599]]
##  6 2 [Cohort 2 [ID's 600 and greater]]
##  7 1 [Cohort 1 [ID's 100-599]]
##  8 2 [Cohort 2 [ID's 600 and greater]]
##  9 2 [Cohort 2 [ID's 600 and greater]]
## 10 2 [Cohort 2 [ID's 600 and greater]]
## # ... with 490 more rows
## # i Use `print(n = ...)` to see more rows
```